

|              |   |
|--------------|---|
| Title        | A Complete Axiomatic Semantics for the CSP Stable-Failures Model  |
| Author(s)    | Isobe, Yoshinao   |
| Citation     |   |
| Issue Date   | 2006-11-30  |
| Type         | Presentation  |
| Text version | publisher   |
| URL          | <a href="http://hdl.handle.net/10119/8308">http://hdl.handle.net/10119/8308</a>   |
| Rights       |   |
| Description  | Theorem Proving and Provers Meeting(2nd TPP)での発表資料, 開催: 2006年11月29日 ~ 30日, 開催場所: JAIST 情報科学研究科棟II・Collaboration Room 7 (5F) |

# A Complete Axiomatic Semantics for the CSP Stable-Failures Model

**Yoshinao Isobe**, AIST, Japan

in cooperation with  
Markus Roggenbach, University of Wales Swansea, UK



TPP 2006 (30/11/2006)

(also see **CONCUR 2006**)

# Overview

## 1 Introduction

- Process algebra
- Motivation

## 2 Non-determinism (ND)

- Syntax of unbounded ND

## 3 Axiom system $\mathcal{A}_F$

- Differences from finite version
- Sequentialisation and Normalisation

## 4 CSP-Prover

- A deep-encoding of CSP in Isabelle

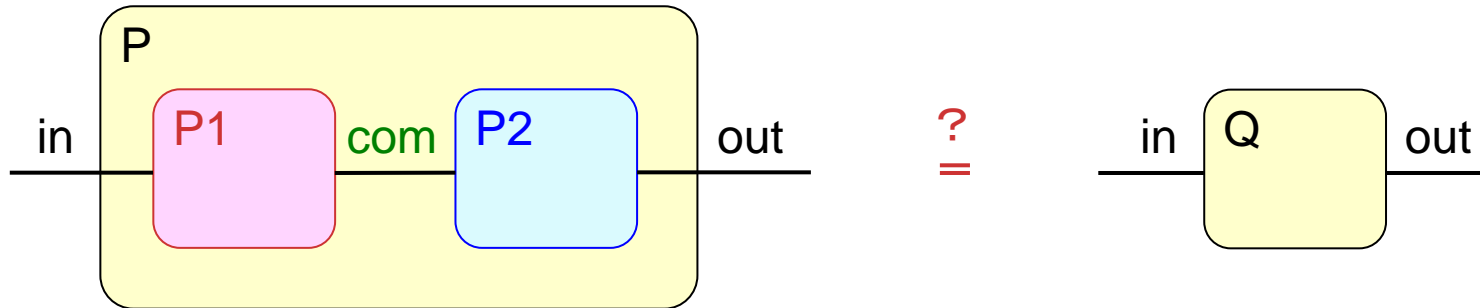
## 5 Conclusion

- Summary and future work

# Introduction

# Process algebra

a formal framework to describe and analyze **concurrent processes**.


$$P = (P1 \parallel [com] \parallel P2) \setminus com$$
$$P1 = in \rightarrow com \rightarrow STOP$$
$$P2 = com \rightarrow out \rightarrow STOP$$
$$P \stackrel{?}{=} Q$$
$$Q = in \rightarrow out \rightarrow STOP$$

## 3 styles of semantics

- Operational semantics
- Denotational semantics
- Axiomatic semantics

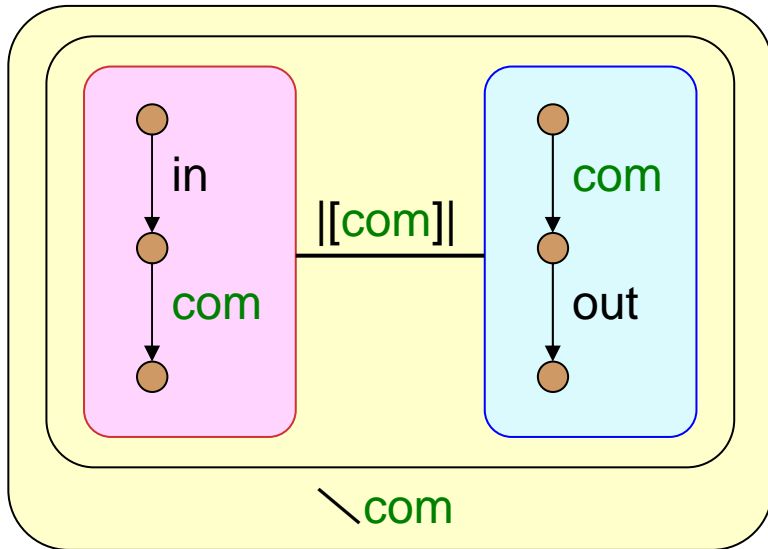
# Operational semantics

$P = (P1 \parallel [com] \parallel P2) \setminus com$

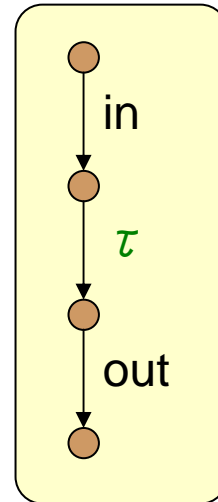
$P1 = in \rightarrow com \rightarrow STOP$

$P2 = com \rightarrow out \rightarrow STOP$

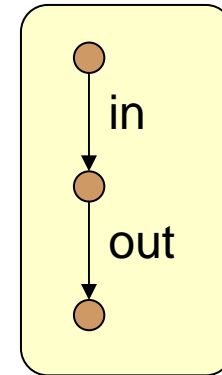
$Q = in \rightarrow out \rightarrow STOP$



=



=



Graph

# Denotational semantics

$$P = (P1 \text{ [[com]] } P2) \setminus \text{com}$$

$$P1 = \text{in} \rightarrow \text{com} \rightarrow \text{STOP}$$

$$P2 = \text{com} \rightarrow \text{out} \rightarrow \text{STOP}$$

Domain  
(traces model)

$$Q = \text{in} \rightarrow \text{out} \rightarrow \text{STOP}$$

$$\text{traces}(Q) = \{ \langle \rangle, \langle \text{in} \rangle, \langle \text{in.out} \rangle \}$$

||

$$\begin{aligned} \text{traces}(P) &= \{ (t_1 \text{ [[com]] } t_2) \setminus \text{com} \mid t_1 \in \text{traces}(P1), t_2 \in \text{traces}(P2) \} \\ &= \{ \langle \rangle, \langle \text{in} \rangle, \langle \text{in.out} \rangle \} \end{aligned}$$

$$\text{traces}(P1) = \{ \langle \rangle, \langle \text{in} \rangle, \langle \text{in.com} \rangle \}$$

$$\text{traces}(P2) = \{ \langle \rangle, \langle \text{com} \rangle, \langle \text{com.out} \rangle \}$$

# Denotational semantics

$$P = (P1 \parallel [com] P2) \setminus com$$

$$P1 = in \rightarrow com \rightarrow STOP$$

$$P2 = com \rightarrow out \rightarrow STOP$$

$$Q = in \rightarrow out \rightarrow STOP$$

Domain  
(stable-failures  
model)

$$traces(Q) = \{ \langle \rangle, \langle in \rangle, \langle in.out \rangle \}$$

$$failures(Q) = \{ (\langle \rangle, \{out\}), (\langle in \rangle, \{in\}), (\langle in.out \rangle, \{in, out\}) \}$$

$$\parallel$$

$$traces(P) = \{ \langle \rangle, \langle in \rangle, \langle in.out \rangle \}$$

$$failures(P) = \{ (\langle \rangle, \{out\}), (\langle in \rangle, \{in\}), (\langle in.out \rangle, \{in, out\}) \}$$

refusals (refused events)



# Axiomatic semantics

$$P = (P1 \text{ [[com]] } P2) \setminus \text{com}$$

$$P1 = \text{in} \rightarrow \text{com} \rightarrow \text{STOP}$$

$$P2 = \text{com} \rightarrow \text{out} \rightarrow \text{STOP}$$

$$\begin{aligned}
 P &= (P1 \text{ [[com]] } P2) \setminus \text{com} \\
 &= ((\text{in} \rightarrow \text{com} \rightarrow \text{STOP}) \text{ [[com]] } (\text{com} \rightarrow \text{out} \rightarrow \text{STOP})) \setminus \text{com} \\
 &= (\text{in} \rightarrow ((\text{com} \rightarrow \text{STOP}) \text{ [[com]] } (\text{com} \rightarrow \text{out} \rightarrow \text{STOP}))) \setminus \text{com} && \text{by [para}_2\text{]} \\
 &= \text{in} \rightarrow ((\text{com} \rightarrow \text{STOP}) \text{ [[com]] } (\text{com} \rightarrow \text{out} \rightarrow \text{STOP})) \setminus \text{com} && \text{by [hide}_2\text{]} \\
 &= \text{in} \rightarrow (\text{com} \rightarrow (\text{STOP} \text{ [[com]] } (\text{out} \rightarrow \text{STOP}))) \setminus \text{com} && \text{by [para}_1\text{]} \\
 &= \text{in} \rightarrow (\text{STOP} \text{ [[com]] } (\text{out} \rightarrow \text{STOP})) \setminus \text{com} && \text{by [hide}_1\text{]} \\
 &= \text{in} \rightarrow (\text{out} \rightarrow (\text{STOP} \text{ [[com]] } \text{STOP})) \setminus \text{com} && \vdots \\
 &= \text{in} \rightarrow \text{out} \rightarrow (\text{STOP} \text{ [[com]] } \text{STOP}) \setminus \text{com} \\
 &= \text{in} \rightarrow \text{out} \rightarrow \text{STOP} \setminus \text{com} \\
 &= \text{in} \rightarrow \text{out} \rightarrow \text{STOP} = Q
 \end{aligned}$$

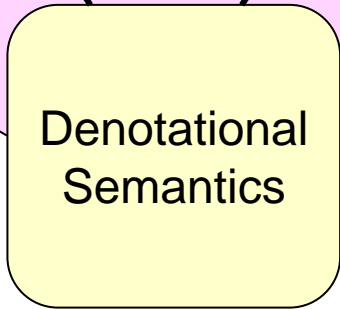
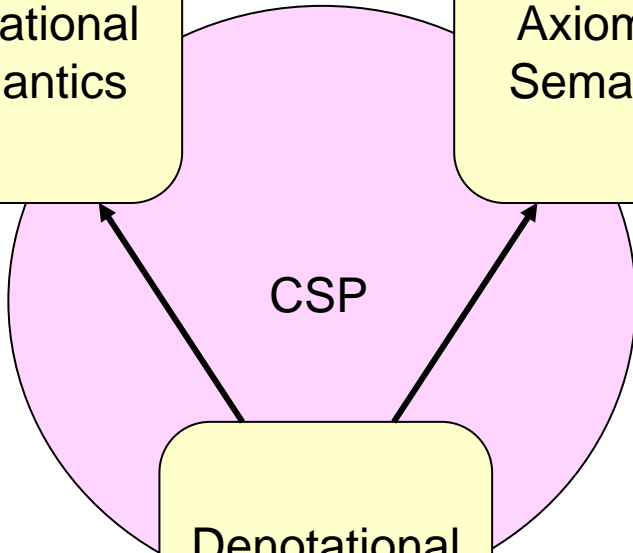
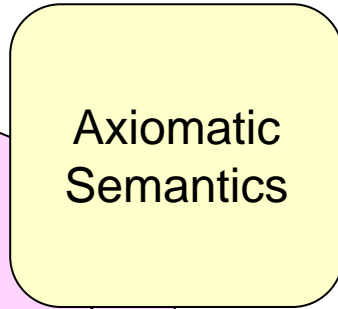
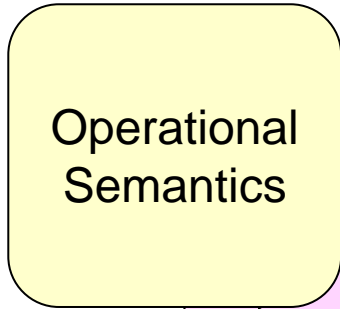
$$Q = \text{in} \rightarrow \text{out} \rightarrow \text{STOP}$$

|               |                      |   |
|---------------|----------------------|---|
| axiom system: | [para <sub>1</sub> ] | $(a \rightarrow P) \text{ [[a]] } (a \rightarrow Q) = a \rightarrow (P \text{ [[a]] } Q)$                 |
|               | [para <sub>2</sub> ] | $(a \rightarrow P) \text{ [[b]] } (b \rightarrow Q) = a \rightarrow (P \text{ [[b]] } (b \rightarrow Q))$ |
|               | [hide <sub>1</sub> ] | $(a \rightarrow P) \setminus a = P \setminus a$   |
|               | [hide <sub>2</sub> ] | $(b \rightarrow P) \setminus a = b \rightarrow (P \setminus a)$   |
|               | ⋮                    |   |

# Process algebra (CSP)

Model checking  
(e.g. FDR)

Theorem proving  
(e.g. **CSP-Prover**)



Definition

open question of CSP

Completeness ?  
for **unbounded** nondeterministic CSP over an **arbitrary** alphabet

c.f. The best known results apply for **finitely** nondeterministic CSP over a **finite** alphabet.  
[Brooks(1983) , Roscoe (1998)]

Our question is:

Is it possible to prove the equality of two CSP-processes by **algebraic laws** without using **denotational semantics**?

Semantics

Definition

C.I.

The best known results apply for **finitely** nondeterministic CSP over a **finite** alphabet.  
[Brooks(1983) , Roscoe (1998)]

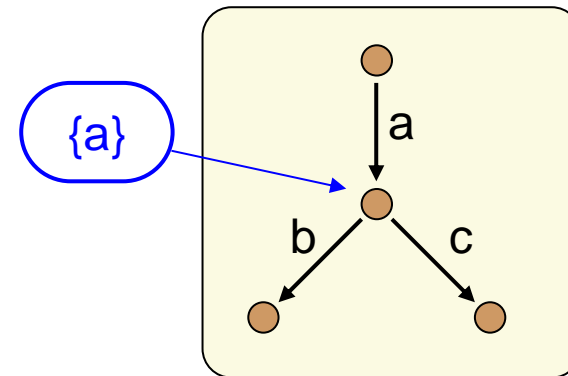
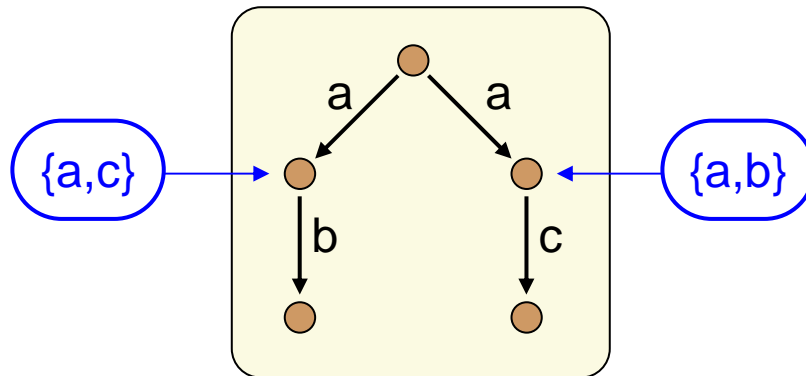
stic  
et

**Non-determinism**

# External choices

external choice  $\square$

$a \rightarrow b \rightarrow \text{STOP} \square a \rightarrow c \rightarrow \text{STOP} \not\equiv_{\mathcal{F}} a \rightarrow (b \rightarrow \text{STOP} \square c \rightarrow \text{STOP})$

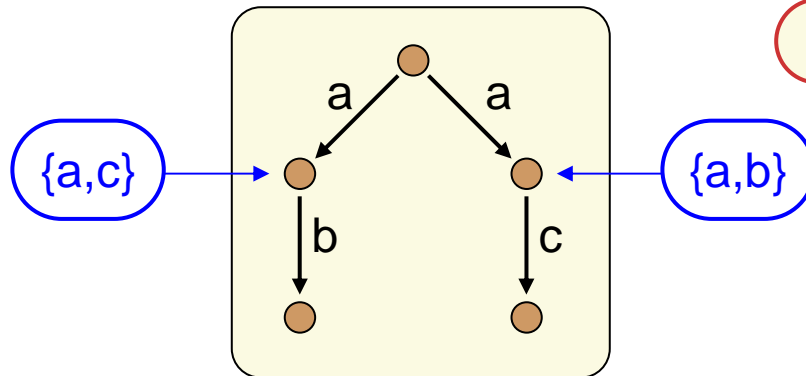


We focus on the **stable-failures model** suitable for describing **infinite systems** and **deadlock** analysis.

# Internal choices

internal choice  $\sqcap$

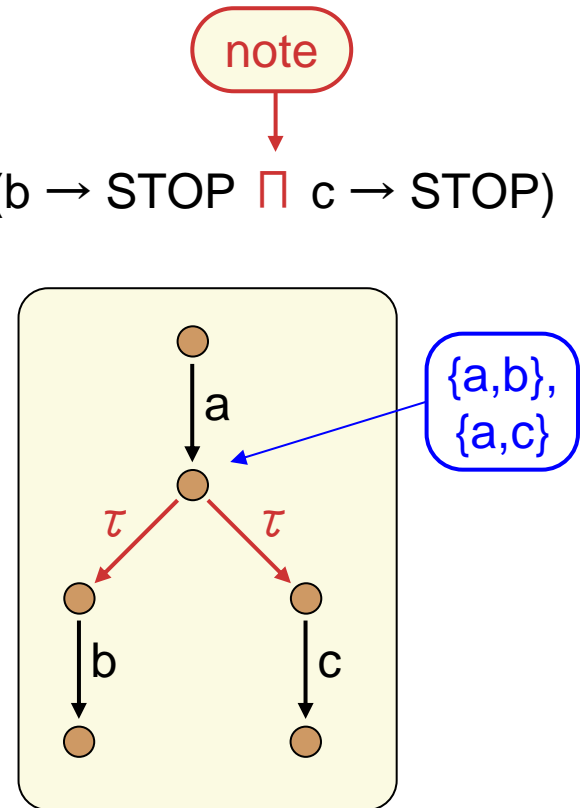
$a \rightarrow b \rightarrow \text{STOP} \sqcap a \rightarrow c \rightarrow \text{STOP}$



$\equiv_{\mathcal{F}}$

note

$a \rightarrow (b \rightarrow \text{STOP} \sqcap c \rightarrow \text{STOP})$



# Unbounded non-determinism

14

## binary internal choice

Random Number Generator  
 $n \in \{0, 1\}$

rand( $n$ )

$\text{RNG} = (\text{rand}(0) \rightarrow \text{STOP}) \sqcap (\text{rand}(1) \rightarrow \text{STOP})$

## general internal choice

Random Number Generator  
 $n \in \text{Nat} = \{0, 1, 2, \dots\}$

rand( $n$ )

$\text{RNG} = \sqcap \{ \text{rand}(n) \rightarrow \text{STOP} \mid n \in \text{Nat} \}$

a set of processes

# Standard CSP

## Syntax

a set of processes

Proc ::= STOP | a → Proc | Proc □ Proc |  $\prod$  (Proc Set) | ...

## Isabelle type

'a : type of alphabet (events)  $\Sigma$

|                                |                     |               |
|--------------------------------|---------------------|---------------|
| <b>datatype</b> 'a proc = STOP |                     |               |
| Act_prefix                     | "'a" "'a proc"      | ( _ → _ )     |
| Ext_choice                     | "'a proc" "'a proc" | ( _ □ _ )     |
| G_Int_choice                   | "'a proc set"       | ( $\prod$ _ ) |
| ...                            |                     |               |

⇒ cardinality mismatch





# CSP<sub>TP</sub>

## Syntax

process function

Proc ::= STOP | a → Proc | Proc □ Proc |  $\prod$  (Proc Fun) | ...

## Isabelle type

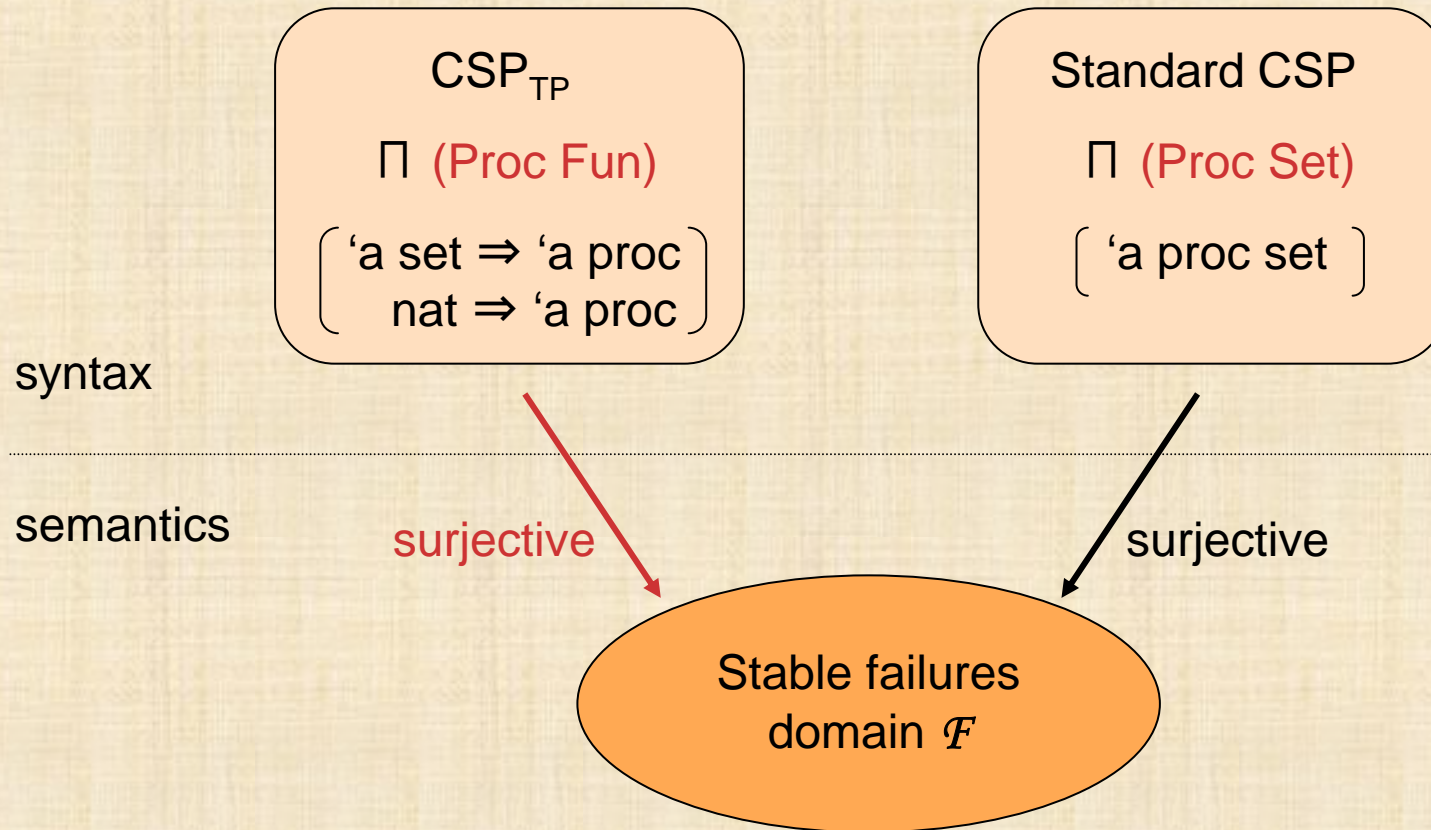
**datatype** 'a proc = STOP

|                |                       |                            |
|----------------|-----------------------|----------------------------|
| Act_prefix     | " 'a" " 'a proc"      | ( _ → _ )                  |
| Ext_choice     | " 'a proc" " 'a proc" | ( _ □ _ )                  |
| Set_Int_choice | " 'a set ⇒ 'a proc"   | ( $\prod_{\text{set}}$ _ ) |
| Nat_Int_choice | " nat ⇒ 'a proc"      | ( $\prod_{\text{nat}}$ _ ) |
| ...            |                       |                            |

note these types

# Relation to 'Standard CSP'

## Expressive power



Div: The **bottom** element  
(in semantic domain)

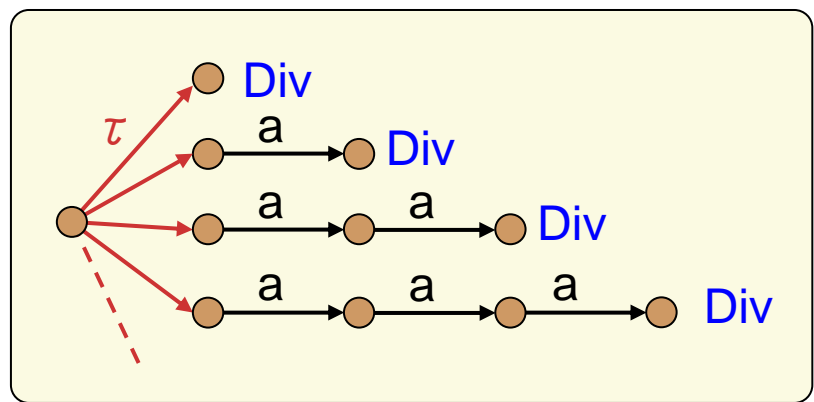
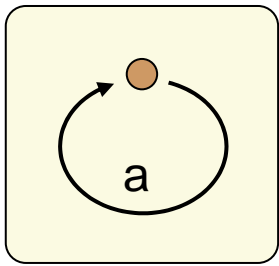
# Recursive processes

$$\text{Loop} = a \rightarrow \text{Loop}$$

Loop

$$\begin{aligned} \text{Loop}^{(0)} &= \text{Div} \\ \text{Loop}^{(n+1)} &= a \rightarrow \text{Loop}^{(n)} \end{aligned}$$

$$=_{\mathcal{F}} \prod \{ \text{Loop}^{(n)} \mid n \in \text{Nat} \}$$



$$=_{\mathcal{F}}$$

Axiom system

# Axiom system $\mathcal{A}_F$

## axiom system $\mathcal{A}_F$

$\mathcal{A}_F$  is sound and complete for the stable failures equivalence over **unbounded** nondeterministic processes with an **arbitrary** alphabet.

$$\forall P, Q \in \text{Proc.} \quad \mathcal{A}_F \vdash P = Q \Leftrightarrow P =_F Q$$

Important **differences** from the standard axioms for finite processes appear in the laws for

- (1) parallel composition in combination with timeout (**corrected**)
- (2) internal choice in combination with Skip (**extended with infinity**)
- (3) depth restriction operator (**new**)

$P \downarrow n$  : depth restriction by the  $n^{\text{th}}$  step

examples

$(\text{Stop}) \downarrow 2 \quad =_{\mathcal{F}} \quad \text{STOP}$

$(a_1 \rightarrow \text{Stop}) \downarrow 2 \quad =_{\mathcal{F}} \quad a_1 \rightarrow \text{STOP}$

$(a_1 \rightarrow a_2 \rightarrow \text{Stop}) \downarrow 2 \quad =_{\mathcal{F}} \quad a_1 \rightarrow a_2 \rightarrow \text{Div}$

$(a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow \text{Stop}) \downarrow 2 \quad =_{\mathcal{F}} \quad a_1 \rightarrow a_2 \rightarrow \text{Div}$

$(a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow a_4 \rightarrow \text{Stop}) \downarrow 2 \quad =_{\mathcal{F}} \quad a_1 \rightarrow a_2 \rightarrow \text{Div}$

all the executions are cut off at the  $2^{\text{nd}}$  step

$$P =_{\mathcal{F}} \prod_{\text{nat}} (\lambda n \bullet (P \downarrow n))$$

# How to normalise

any process

key point :  
remove **Hiding operators** by a **function recursively**  
defined on the **process structure**.

full sequential form

key point :  
normalise  $((\prod_{\text{set}} P(X)) \downarrow n)$  by a **function recursively**  
defined on the depth **n**.

(extended) full normal form

**note**

Induction on the process structure **cannot** be  
applied to a **family** of processes  $P(X)$  ( $X \subseteq \Sigma$ )

# How to normalise

any p

$$\forall X \subseteq \Sigma. P(X) \in \text{FNF}$$

$\prod_{\text{set}} P(X)$  can be normalized if  $\Sigma$  is finite.

However,  $\Sigma$  can be infinite!

full sequential form

key point :

normalise  $((\prod_{\text{set}} P(X)) \downarrow n)$  by a function recursively defined on the depth  $n$ .

(extended) full normal form

note

Induction on the process structure **cannot** be applied to a family of processes  $P(X)$  ( $X \subseteq \Sigma$ )



# How to normalise

note

$$P \downarrow n =_F Q \downarrow n \not\Rightarrow (P \setminus X) \downarrow n =_F (Q \setminus X) \downarrow n$$

any process

key point :  
remove **Hiding operators** by a **function recursively**  
defined on the **process structure**.

full sequential form

key point :  
normalise  $((\prod_{\text{set}} P(X)) \downarrow n)$  by a **function recursively**  
defined on the **depth n**.

(extended) full normal form

note

Induction on the process structure **cannot** be  
applied to a **family** of processes  $P(X)$  ( $X \subseteq \Sigma$ )

## Full Sequential Form (FSF)

FSF contains only “**sequential**” operators such as  $\square$ ,  $!!$ , and Stop.

---

The following function  $\text{Seq}: \text{Proc} \rightarrow \text{FSF}$  can be **recursively** defined over the process **structure**.

### Theorem 3

$$\forall P \in \text{Proc}. \mathcal{A}_F \vdash P = \text{Seq}(P)$$

This theorem can be proven by **structural induction on P**.

### note

The sequential process  $\text{Seq}(P)$  **cannot** be necessarily automatically computed because  $\text{Seq}(P)$  often needs infinite computations, for example

$$\text{Seq}(\prod s \bullet P(s))$$

requires to compute  $\text{Seq}(P(s))$  for all  $s \in S$ , where  $S$  may be infinite.

## Full Normal form

### Syntactic equality?

$$\prod s \bullet (\prod s' \bullet P_{\text{seq}}(s, s')) \in \text{FSF}$$

$$\prod s' \bullet (\prod s \bullet P_{\text{seq}}(s, s')) \in \text{FSF}$$

(semantically equal but  
syntactically different)

### Full Normal Form (FNF) (similar to the standard FNF)

FNF is a more specialized form than FSF, for giving the syntactic equality.

#### Theorem 4

$$\forall P, Q \in \text{FNF}. P =_F Q \Leftrightarrow P \equiv Q \quad (\text{syntactic equality})$$

# Full Normal form

The following function  $\text{Norm}: \text{FSF} \rightarrow \text{FNF}$  can be **recursively** defined on **the depth  $n$**  and the structure over **FSF**.

Lemma 2

$$\forall P \in \text{FSF}. \mathcal{A}_{\mathcal{F}} \vdash P \downarrow n = \text{Norm}_{(n)}(P)$$

This theorem can be proven by the induction on  $n$  and **structural** induction on  $P$ .

$P$  may be  $(!! s:S \bullet P'(s))$

FNF does not capture all processes

There is **no function**  $\text{Norm}'$  such that  $\forall P \in \text{FSF}. \mathcal{A}_{\mathcal{F}} \vdash P = \text{Norm}'(P)$

Theorem 5

$$\exists P \in \text{FSF}. \forall P' \in \text{FNF}. P \not\equiv_{\mathcal{F}} P'$$

reminder

## Extended Full Normal Form

$$P =_{\mathcal{F}} \prod n \bullet (P \downarrow n)$$

## Extended Full Normal Form (XFNF)

$$P = \prod n \bullet P'(n) \quad \text{if} \quad \left( \begin{array}{l} (1) \quad \forall n. P'(n) \in \text{FNF} \text{ and} \\ (2) \quad \forall n. P \downarrow n =_{\mathcal{F}} P'(n) \end{array} \right)$$

infinite internal choice over fully normalised processes for finite depths

## Theorem 6

$$\forall P, Q \in \text{XFNF}. \quad P =_{\mathcal{F}} Q \quad \Leftrightarrow \quad P \equiv Q \quad (\text{syntactic equality})$$

## Theorem 7

$$\forall P \in \text{Proc}. \quad \exists P' \in \text{XFNF}. \quad \mathcal{A}_{\mathcal{F}} \vdash P = X_{\text{norm}}(P')$$

$$X_{\text{norm}}(P) \equiv \prod n \bullet (\text{Norm}_{(n)}(\text{Seq}(P)))$$

# Completeness

## Corollary

$$\forall P, Q \in \text{Proc. } P =_{\mathcal{F}} Q \Rightarrow \mathcal{A}_{\mathcal{F}} \vdash P = Q$$

Let  $P =_{\mathcal{F}} Q$ , then

$$\mathcal{A}_{\mathcal{F}} \vdash P = X_{\text{norm}}(P) \equiv X_{\text{norm}}(Q) = Q$$

## Theorem 6

$$\forall P, Q \in \text{XFNF. } P =_{\mathcal{F}} Q \Leftrightarrow P \equiv Q \quad (\text{syntactic equality})$$

## Theorem 7

$$\forall P \in \text{Proc. } \exists P' \in \text{XFNF. } \mathcal{A}_{\mathcal{F}} \vdash P = X_{\text{norm}}(P)$$

$$X_{\text{norm}}(P) \equiv \prod n \bullet (\text{Norm}_{(n)}(\text{Seq}(P)))$$

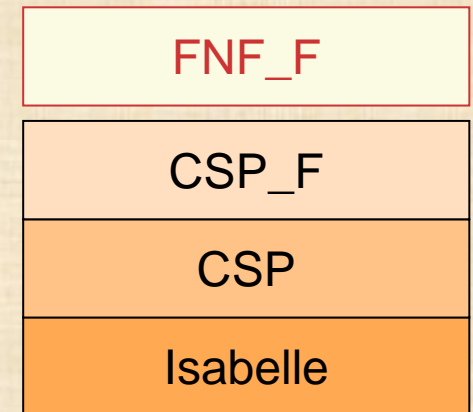
CSP-Prover

# CSP-Prover

**CSP-Prover:** a deep encoding of CSP in the generic theorem prover Isabelle

**includes** fixed point theorems, definitions of syntax and semantics, CSP-laws, semi-automatic proof tactics, etc.

- Verification of infinite state systems  
e.g. EP2 (an electronic payment system)
- Establishing new theorems on CSP  
e.g. Soundness and completeness of  $\mathcal{A}_F$



- References: 1. Y.Isobe and M.Roggenbach, A Generic Theorem Prover of CSP refinement, **TACAS 2005**, LNCS 3440, pp.108-123, 2005
2. Y.Isobe and M.Roggenbach, A complete axiomatic semantics for CSP stable failures model, **CONCUR 2006**, LNCS 4237, pp.158-172, 2006

Web-site: <http://staff.aist.go.jp/y-isobe/CSP-Prover/CSP-Prover.html>



**Conclusion**

# Summary and Future Work

## Summary

1. **Complete** axiomatic semantics of the stable failures model
2. Our CSP dialect is **expressive** with respect to the stable failures model
3. Implementation & Verification of all results in **CSP-Prover**
4. **Correction** of two well-known step laws

The errors as well as our corrections have been approved by Bill Roscoe, Oxford.

## Future work

1. Improve **proof tactics** in CSP-Prover based on the normal forms
2. Develop completeness results for **other CSP models**